МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Системного программирования»

Дисциплина «Операционные системы»

Отчет по курсовой работе

Выполнил:

студент группы БСТ2104

Мажукин И.Н.

Проверила: Алексанян Д. А.

Москва, 2023 г.

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Разработать многозадачное приложение, выполняющее получение, сбор и вывод системной информации в соответствии с вариантом задания (таблица 1). Номер варианта задания на курсовую работу совпадает с номером, под которым записана фамилия студента в журнале группы.

Таблица 1 – Индивидуальный вариант задания

|  |  |  |
| --- | --- | --- |
| Вариант | Сервер 1 | Сервер 2 |
| получает от клиента/выполняет и возвращает в качестве результата | |
| 7 | * название используемого видеоадаптера * размер клиентской области | * количество свободных байтов файла подкачки * объем используемой физической памяти в единицах, переданных клиентом (в байтах, мегабайтах, гигабайтах по выбору пользователя) |

**Дополнительные задания**

1. Реализация графического интерфейса на стороне клиента.
2. При периодическом обновлении информации по запросу от клиента (организовать выбор режима отправки периодических запросов к серверу в рамках текущей сессии) новые данные передается сервером в ответ только в том случае, если что-либо изменилось с момента прошлого обновления информации для данного клиента.
3. Наличие версии сервера или клиента для другой ОС (т.е. если клиент и сервер могут работать под разными ОС).

**СОДЕРЖАНИЕ**

[**Введение** 4](#_Toc154089298)

[**1. Постановка задачи** 5](#_Toc154089299)

[**2. Теоретические основы разрабатываемой темы** 6](#_Toc154089300)

[**3. Разработка и описание алгоритма работы приложения** 7](#_Toc154089301)

[**4. Результат работы программы** 12](#_Toc154089302)

[**Заключение** 16](#_Toc154089303)

[**Список использованных источников** 16](#_Toc154089304)

[**Приложения** 17](#_Toc154089305)

# **Введение**

В современном мире, где технологии проникают во все сферы нашей жизни, передача и обработка данных становится неотъемлемой частью нашей повседневной деятельности. От промышленных комплексов до систем автоматизации торговли — все они тесно переплетены с необходимостью эффективного обмена информацией.

В этом контексте использование сокетов в клиент-серверных приложениях представляет собой элегантное и востребованное решение. Эти приложения выступают в роли моста, обеспечивающего постоянное взаимодействие между различными процессами. Сервер, выступая поставщиком услуг, готов ожидать запросов от клиентов, предоставляя им необходимые данные или решения вычислительных задач. Клиент, в свою очередь, будучи инициатором запроса, обладает гибкостью подключаться к серверу в любой момент, повышая тем самым эффективность обмена информацией.

Цельюданной курсовой работы является закрепление и углубление теоретических знаний в области современных операционных систем, приобретение практических навыков разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

# **Постановка задачи**

Основной задачей данной курсовой работы является разработка многозадачного приложения, выполняющего получение, сбор и вывод системной информации в соответствии с индивидуальным вариантом задания.

Многозадачное приложение реализуется на основе технологии «клиент-сервер» средствами интегрированной среды разработки Visual Studio Code на языке программирования Python3. Процессы 1 и 2 выступают как серверы приложений, третий процесс является их клиентом. Обмен данными между клиентами и серверами осуществляется при помощи сокетов.

Многозадачное приложение реализуется на основе технологии «клиент-сервер». Два сервера должны поддерживать многопоточность, т.е. обеспечивать одновременное подключение нескольких клиентов. Каждый сервер должен обрабатывать запросы клиентов, предоставляя системную информацию в соответствии с вариантом задания, и передавать её клиенту с указанием времени. Также необходимо разработать клиентскую часть с интерфейсом, позволяющим выбирать сервер для подключения, с возможностью раздельного получения и отображения данных по каждому из пунктов индивидуального задания.

# **2. Теоретические основы разрабатываемой темы**

1. Архитектура «клиент-сервер»:

Архитектура «клиент-сервер» - это модель взаимодействия компьютерных сетей, при которой на одном конце соединения находится клиент, а на другом - сервер. Сервер предоставляет услуги или ресурсы, а клиент их запрашивает.

**Плюсы:**

* **Эффективность**: архитектура «клиент-сервер» позволяет эффективно использовать ресурсы сервера, так как клиент может быть бездействующим, пока сервер обрабатывает запрос.
* **Масштабируемость**: архитектура «клиент-сервер» легко масштабируется, так как можно добавить больше серверов для обработки запросов от большего числа клиентов.
* **Гибкость**: архитектура «клиент-сервер» позволяет использовать различные протоколы и технологии для обмена данными между клиентом и сервером, что делает ее гибкой для различных приложений.

**Минусы**:

* **Надежность**: если сервер не работает, все клиенты не смогут получить доступ к ресурсам или услугам, которые предоставляет сервер. В случае отказа основного сервера, необходимо наличие резервного сервера.
* **Безопасность**: архитектура «клиент-сервер» может быть уязвима для атак, так как клиент может отправлять запросы на сервер, которые могут содержать вредоносный код или попытки взлома.

2. Системные средства коммуникации и синхронизации процессов

* **Сокеты.** Сокет — это средство связи, позволяющее разрабатывать «клиент-сервер» системы для локального или сетевого использования.
* **Многопоточность**: архитектура «клиент-сервер» позволяет обрабатывать несколько клиентских запросов одновременно, используя отдельные потоки на сервере.
* **Протоколы передачи данных**: для обмена данными между клиентом и сервером используются протоколы TCP/IP, которые обеспечивают надежность и управление соединением (TCP) и маршрутизацию и доставку пакетов данных (IP).

# **3. Разработка и описание алгоритма работы приложения**

«Клиент-сервер» система состоит из 5 файлов расширением .py: первый сервер (server\_1), второй сервер (server\_2), клиент (client), файлы для графического изображения серверов (server\_design) и для клиента (client\_desing). Первый и второй серверы имеют один и тот же HOST = "192.168.1.7", но разные порты (для сервера 1 PORT = 2233, для сервера 2 PORT = 2234). После запуска серверов, они ожидают входящее соединение.

После установления соединения клиент выбирает сервер, к которому он хочет подключиться и при этом клиент, отправляя запрос на выбранный сервер с помощью сокетов, возвращает на экран данные от выбранного сервера.

Рассмотрим теперь запросы, которые может пользователь запросить от каждого сервера:

Сервер 1:

* **Вывод название используемого видеоадаптера:**

Клиент, подключившись к первому серверу, передаёт названия видеокарт.

* **Размер клиентской области:** Клиент, подключившись к первому серверу, передаёт размер клиентской области.

Сервер 2:

* **Количество свободных байтов файла подкачки:** Клиент, подключившись ко второму серверу, получает информацию о количестве свободных байтов файла подкачки сервера.
* **объем используемой физической памяти в единицах, переданных клиентом (в байтах, мегабайтах, гигабайтах по выбору пользователя):** Клиент, подключившись ко второму серверу, получает информацию об объеме используемой физической памяти сервера в единицах, которые пользователь выбирает из списка в графическом окне.

Рассмотрим теперь модули, функции и классы, которые были использованы при разработке серверов и клиентов:

**Модули**:

Для сервера 1 и 2:

* **socket** – используется для создания сетевого соединения и обмена данными между клиентами и сервером;
* **threading** – применяется для организации многозадачности и обработки каждого клиента в отдельном потоке;
* **sys** – позволяет управлять системными ресурсами, работать с аргументами командной строки и другими компонентами, связанными с выполнением программы;
* **os** – предоставляет функционал для взаимодействия с операционной системой, например, определения приоритета процесса.
* **datetime** - позволяет выполнять операции с датами и временными интервалами, форматировать дату и время, извлекать информацию о текущей дате и времени.
* **PyQt5.QtCore, PyQt5.QtWidgets, PyQt5.QtGui** - модули являются частью фреймворка PyQt5, который предоставляет возможности для создания графического пользовательского интерфейса (GUI) в приложениях Python. PyQt5.QtCore содержит основные классы и функции для работы с событиями, таймерами и потоками. PyQt5.QtWidgets предоставляет готовые виджеты для построения UI, такие как кнопки, текстовые поля, окна и диалоги. PyQt5.
* **server\_desing** - пользовательский модуль содержит класс Ui\_MainWindow, который определяет графический интерфейс главного окна сервера приложения.

Для клиента:

* **socket** – используется для создания сетевого соединения и обмена данными между клиентами и сервером;
* **threading** – применяется для организации многозадачности и обработки каждого клиента в отдельном потоке;
* **sys** – позволяет управлять системными ресурсами, работать с аргументами командной строки и другими компонентами, связанными с выполнением программы;
* **time** - позволяет выполнять операции с датами и временными интервалами, форматировать дату и время, извлекать информацию о текущей дате и времени.
* **PyQt5.QtWidgets** - модуль является частью фреймворка PyQt5, который предоставляет возможности для создания графического пользовательского интерфейса (GUI) в приложениях Python. PyQt5.QtWidgets предоставляет готовые виджеты для построения UI, такие как кнопки, текстовые поля, окна и диалоги. PyQt5.
* **client\_desing** - пользовательский модуль содержит класс Ui\_MainWindow, который определяет графический интерфейс главного окна сервера приложения
* **enum -** Модуль enum содержит в себе тип для перечисления значений с возможностью итерирования и сравнения**.**

**Функции:**

Для сервера 1 и 2:

ServerThread – это класс-поток, который расширяет функциональность базового класса Thread. Он служит для создания отдельного потока, в котором будет выполняться серверная логика.

\_\_init\_\_(self, window) – конструктор класса, принимающий окно (window) в качестве параметра.

run() содержит основную логику сервера. Устанавливается соединение через сокет, привязывается к определенному хосту и порту, ожидает подключения клиентов с помощью accept() и при каждом новом подключении создается экземпляр потока (Thread) для обработки этого подключения функцией handle\_connection.

handle\_connection(sock, addr, window) – это функция, обрабатывающая подключение клиента, sock - сокет, связанный с клиентом, addr - адрес клиента, window - окно GUI, используемое для отображения информации. Эта функция также обеспечивает обработку входящих данных от клиента, отправку ответа клиенту и управление ими через сокет.

ServerThread.run() – Создает серверный сокет, привязывается к определенному хосту и порту. Ожидает подключения клиентов и запускает обработку подключения в отдельных потоках. Обрабатывает исключение KeyboardInterrupt для остановки сервера. Закрывает серверный сокет по завершении.

mywindow.\_\_init\_\_() – Инициализирует главное окно QMainWindow. Настраивает пользовательский интерфейс из файла server\_design. Устанавливает заголовок окна.

mywindow.addItem(str, data) – Добавляет новый элемент в QListWidget.Объединяет текст str и data и добавляет это в QListWidget. Функция предназначена для отображения информации в списке виджета (QListWidget), позволяя добавлять элементы из других частей программы.

Для клиента:

checkUpdateTimer() – Запускает бесконечный цикл, который проверяет статус соединения с сервером. Если таймер обновления не выключен и соединение с сервером активно, то вызывает соответствующий метод server\_1\_Btn\_click() или server\_2\_Btn\_click() для обновления информации с сервера.

check\_server(ip, port) – Проверяет доступность сервера, пытаясь установить соединение с заданным IP-адресом и портом. Создает сокет, который пытается установить соединение с сервером.

ClientThread.\_\_init\_\_(self, window, serverType) – конструктор класса, принимающий объект окна (window) и тип сервера (serverType), с которым устанавливается соединение. Сохраняет переданные параметры для последующего использования внутри экземпляра класса.

ClientThread.run() – Метод запускается при вызове start() для потока. Инициализирует соединение с сервером по указанным HOST и PORT для указанного типа сервера. Устанавливает sockStatus в 1 для указанного сервера, чтобы указать, что соединение установлено. Вызывает метод receiveFunc() для получения данных от сервера. Обрабатывает исключение KeyboardInterrupt для корректного завершения работы при прерывании пользователем. По завершении, закрывает сокет.

receiveFunc() – Метод является частью класса ClientThread, отвечающий за прием данных от сервера в текущем потоке. Он предназначен для непрерывного приема и обработки данных, пока соединение с сервером открыто и данные поступают.

mywindow – класс, который включает обработчики событий для графического интерфейса приложения, представленного с помощью Ui\_MainWindow, и управляет взаимодействием пользователя с клиентской частью программы.

\_\_init\_\_(self) – Инициализирует окно приложения на основе файла client\_design. Назначает обработчики событий кнопок и элементов интерфейса для отправки запросов на серверы и установки таймера обновления.

Методы server\_1\_Btn\_click, server\_2\_Btn\_click, server\_3\_Btn\_click, server\_4\_Btn\_click - отправляют соответствующие запросы на серверы при нажатии кнопок. При отсутствии подключения пытаются установить его и повторно отправить запрос. Выводят информацию о статусе отправки в графический интерфейс.

Метод on\_combobox\_changed – обработчик выпадающего списка с вариантами выбора единиц измерения данных для задания 2. Вызывается при изменении выбранного элемента в QComboBox, возвращает выбранный текст из QComboBox.

Метод addItem – добавляет элементы в соответствующие списки (listWidget, listWidget\_2) графического интерфейса с указанием их происхождения (сервер 1 или сервер 2). Выводит информацию в консоль для отслеживания.

Методы onAction\_1\_Clicked, onAction\_2\_Clicked, onAction\_3\_Clicked, onAction\_4\_Clicked, onAction\_5\_Clicked – устанавливают таймер обновления на разные временные интервалы при выборе соответствующих опций в меню.

# **4. Результат работы программы**

Результаты работы приложения представлены на рисунках 1-9:

Запускаем клиент в терминале VSC. Результат представлен на рисунке 1

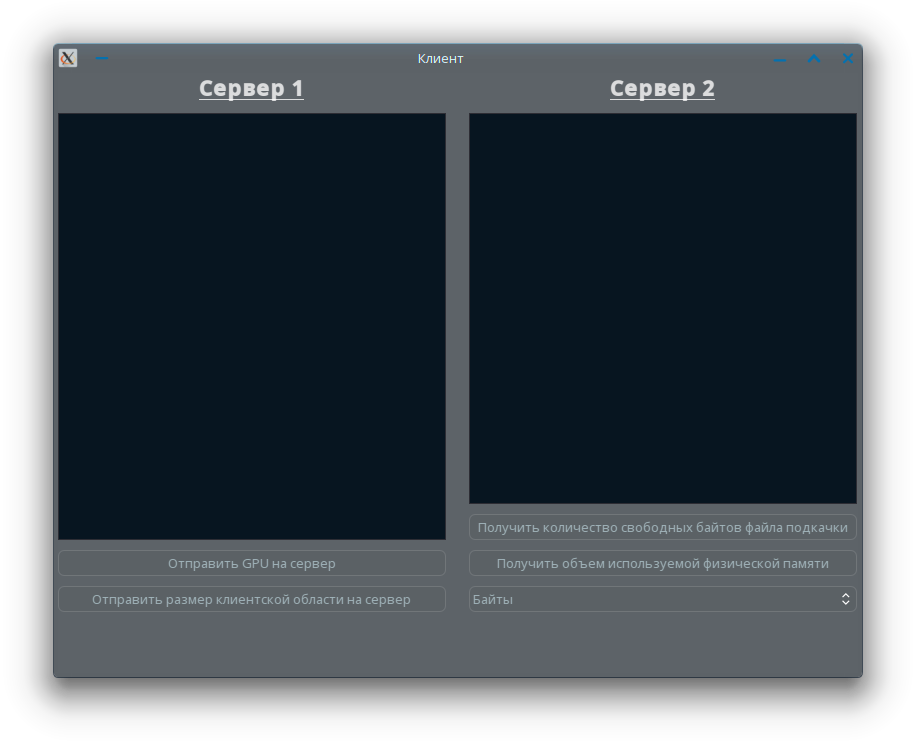


Рисунок 1 – Запуск клиента

Запускаем сервер 1 и выводим информацию об используемых видеоадаптерах

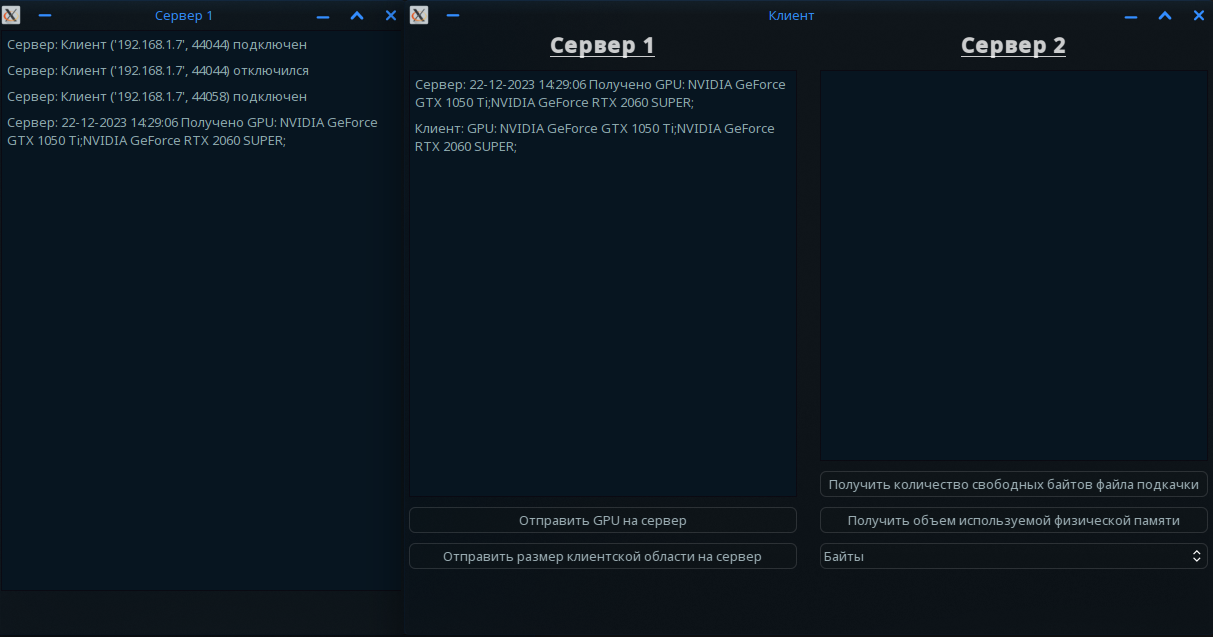


Рисунок 2 – Запуск сервера 1 и вывод необходимой информации

Передадим информацию о размере клиентской области.

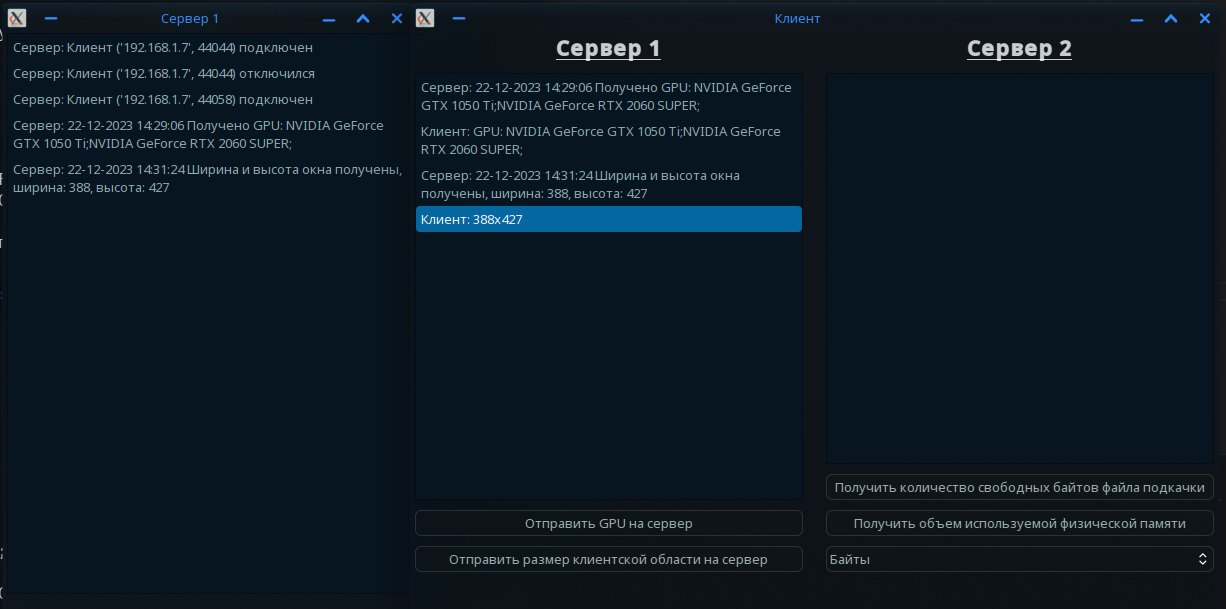


Рисунок 3 – информацию о размере клиентской области

Запускаем сервер 2 и выводим информацию о количестве свободных байтов файла подкачки.

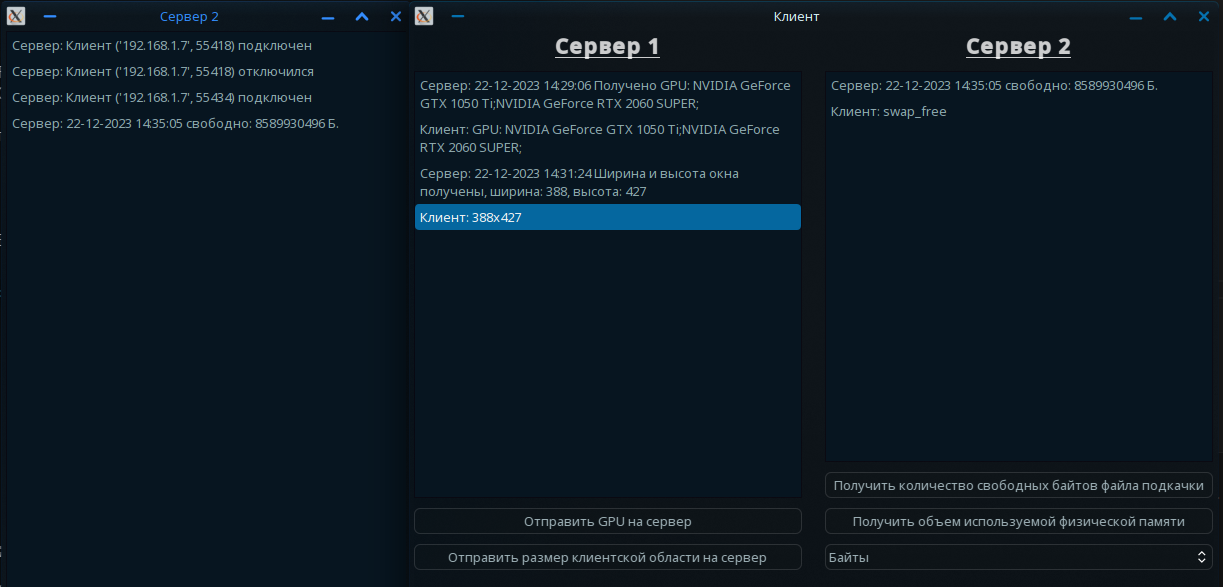


Рисунок 4 – Запуск сервера 2 и вывод свободных байтов файла подкачки.

Запускаем сервер 2 и выводим информацию об объёме используемой физической памяти сервера. Меняем единицы измерения и смотрим на результат.

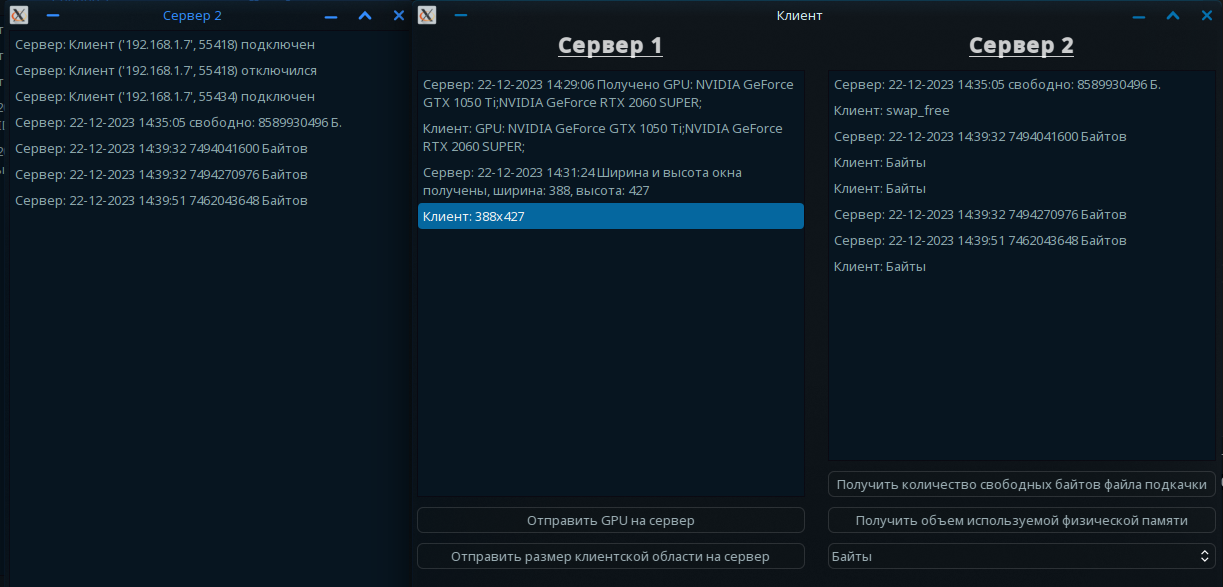


Рисунок 5 – Вывод использования физической памяти в байтах

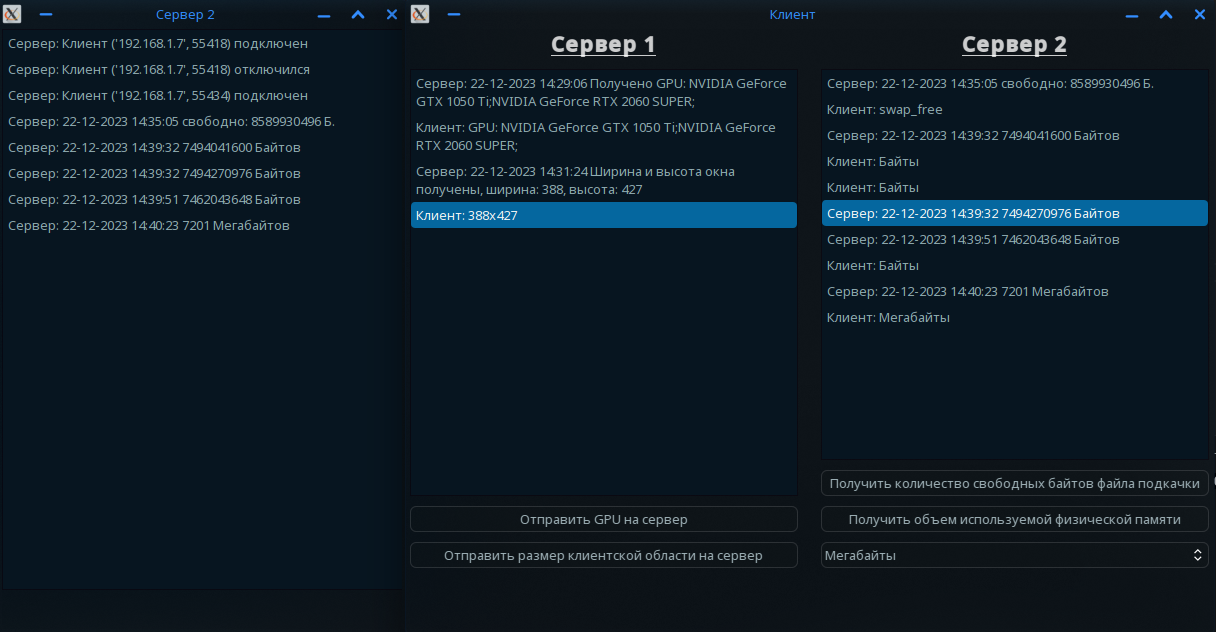


Рисунок 6 – Вывод использования физической памяти в мегабайтах

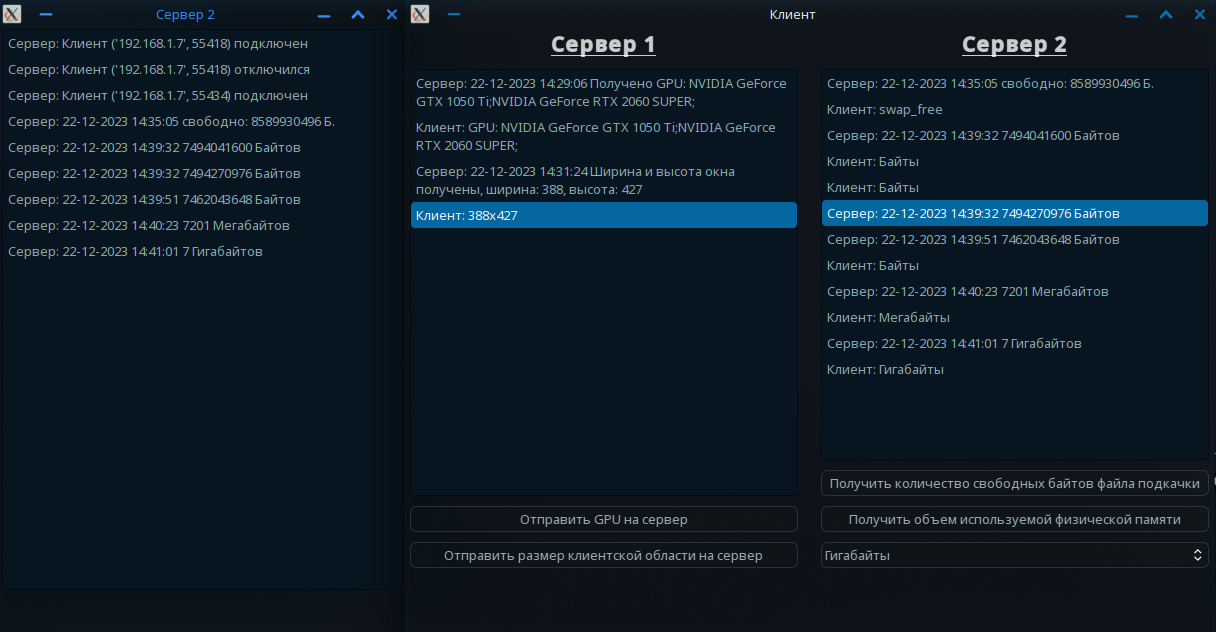


Рисунок 7 – Вывод использования физической памяти в гигабайтах

# **Заключение**

В результате выполнения данной курсовой работы было разработано клиент-серверное приложение, выполняющее получение, сбор и вывод системной информации» в соответствии с индивидуальным вариантом задания. В ходе работы также были закреплены и углублены теоретические знания в области современных операционных систем, приобретены практические навыки разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

# **Список использованных источников**

1. Э.Таненбаум. Современные операционные системы. 3-изд. – СПб.: Питер, 2011
2. Э.Таненбаум, Х.Бос. Современные операционные системы. 4-изд. – СПб.: Питер, 2015
3. В.Г.Олифер, Н.А.Олифер. Сетевые операционные системы. – СПб.: Питер, 2009
4. С.В. Назаров, А.И. Широков. «Современные операционные системы». — М., Бином, 2013
5. Марк Руссинович, Дэвид Соломон, Алекс Ионеску, Павел Йосифович. Внутреннее устройство Windows.7-е издание . - СПб.: Питер, 2018
6. Аллан Фарингтон «Windows 11: полное руководство пользователя новой операционной системы». — Издательский отдел факультета ВМиК МГУ, 2008

# **Приложения**

Ниже представлен код клиента(client.py):

#!/usr/bin/env python3

import sys, socket, time

from PyQt5 import QtWidgets

from client\_design import Ui\_MainWindow # импорт сгенерированного файла

from threading import Thread, Lock

from enum import Enum

import GPUtil

def get\_gpu\_info() :

try:

# Получение списка доступных устройств GPU

gpus = GPUtil.getGPUs()

gpu\_info = []

for gpu in gpus:

gpu\_info.append({

'ID': gpu.id,

'Name': gpu.name,

'Driver': gpu.driver,

'Memory Total': f"{gpu.memoryTotal} MB",

'Memory Used': f"{gpu.memoryUsed} MB",

'Memory Free': f"{gpu.memoryFree} MB"

})

return gpu\_info

except Exception as e:

print(f"Ошибка при получении информации о GPU: {e}")

return None

class UpdatePeriod(Enum):

OFF = 0

ONE\_MIN = 1

FIVE\_MIN = 5

FIFETEEN\_MIN = 15

ONE\_HOUR = 60

HOST = "192.168.1.7"

PORT = [2233, 2234]

BUF\_SIZE = 1024 # Размер буфера для получения сообшения

sock = [None, None] # Сокеты

sockStatus = [0, 0] # Статусы подключения к серверам

IS\_RECONNECT\_ENABLED = False # Флаг для переподключения при падении клиента

updateTimer = UpdatePeriod.OFF.value # Таймер обновления информации

class mywindow(QtWidgets.QMainWindow):

def \_\_init\_\_(self):

super(mywindow, self).\_\_init\_\_()

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

# Обработка нажатий кнопок

self.ui.sendToServer\_1.clicked.connect(self.server\_1\_Btn\_click)

self.ui.sendToServer\_2.clicked.connect(self.server\_2\_Btn\_click)

self.ui.sendToServer\_3.clicked.connect(self.server\_3\_Btn\_click)

self.ui.sendToServer\_4.clicked.connect(self.server\_4\_Btn\_click)

# Обработка нажатий меню выбора таймера обновления

self.ui.action\_1.triggered.connect(self.onAction\_1\_Clicked)

self.ui.action\_2.triggered.connect(self.onAction\_2\_Clicked)

self.ui.action\_3.triggered.connect(self.onAction\_3\_Clicked)

self.ui.action\_4.triggered.connect(self.onAction\_4\_Clicked)

self.ui.action\_5.triggered.connect(self.onAction\_5\_Clicked)

self.ui.combo\_box.currentIndexChanged.connect(self.on\_combobox\_changed)

# Отправка на сервер 1

def server\_1\_Btn\_click(self):

global sock, sockStatus

data = "GPU: "

# Получение информации о видеоадаптере

gpu\_info = get\_gpu\_info()

if gpu\_info:

for gpu in gpu\_info:

data += gpu['Name'] + ";"

# Если сервер уже подключен

if(sockStatus[0] != 0):

try:

# Отправка

sock[0].sendall(data.encode())

print(data)

self.ui.listWidget.addItem('Клиент: ' + data)

except OSError:

data = 'Ошибка! Сервер 1 не подключен!'

print(data)

sockStatus[0] = 0

self.ui.listWidget.addItem('Клиент: ' + data)

# Если сервер еще не подключен

else:

# Пробуем подключиться

if(check\_server(HOST, PORT[0]) == True):

clientThread = ClientThread(window, 0)

clientThread.start()

time.sleep(0.5)

self.server\_1\_Btn\_click()

# Если не удалось подключиться

else:

data = 'Ошибка! Сервер 1 не подключен!'

print(data)

# Вывод в графический интерфейс

self.ui.listWidget.addItem('Клиент: ' + data)

def server\_3\_Btn\_click(self):

global sock, sockStatus

# получение ширины и высоты окна

data = str(self.ui.listWidget.geometry().width())

data += 'x' + str(self.ui.listWidget.geometry().height())

# Если сервер уже подключен

if(sockStatus[0] != 0):

try:

# Отправка

sock[0].sendall(data.encode())

print(data)

self.ui.listWidget.addItem('Клиент: ' + data)

except OSError:

data = 'Ошибка! Сервер 1 не подключен!'

print(data)

sockStatus[0] = 0

self.ui.listWidget.addItem('Клиент: ' + data)

# Если сервер еще не подключен

else:

# Пробуем подключиться

if(check\_server(HOST, PORT[0]) == True):

clientThread = ClientThread(window, 0)

clientThread.start()

time.sleep(0.5)

self.server\_3\_Btn\_click()

# Если не удалось подключиться

else:

data = 'Ошибка! Сервер 1 не подключен!'

print(data)

# Вывод в графический интерфейс

self.ui.listWidget.addItem('Клиент: ' + data)

# Отправка на сервер 2

def server\_2\_Btn\_click(self):

global sockStatus

data = 'swap\_free'

# Если сервер уже подключен

if(sockStatus[1] != 0):

try:

# Отправка

sock[1].sendall(data.encode())

print(data)

self.ui.listWidget\_2.addItem('Клиент: ' + data)

except OSError:

data = 'Ошибка! Сервер 2 не подключен!'

print(data)

sockStatus[1] = 0

self.ui.listWidget\_2.addItem('Клиент: ' + data)

# Если сервер еще не подключен

else:

# Пробуем подключиться

if(check\_server(HOST, PORT[1]) == True):

clientThread = ClientThread(window, 1)

clientThread.start()

time.sleep(0.5)

self.server\_2\_Btn\_click()

# Если не удалось подключиться

else:

data = 'Ошибка! Сервер 2 не подключен!'

print(data)

# Вывод в графический интерфейс

self.ui.listWidget\_2.addItem('Клиент: ' + data)

# свободная память

def server\_4\_Btn\_click(self):

global sockStatus

data = self.on\_combobox\_changed()

# Если сервер уже подключен

if(sockStatus[1] != 0):

try:

# Отправка

sock[1].sendall(data.encode())

print(data)

self.ui.listWidget\_2.addItem('Клиент: ' + data)

except OSError:

data = 'Ошибка! Сервер 2 не подключен!'

print(data)

sockStatus[1] = 0

self.ui.listWidget\_2.addItem('Клиент: ' + data)

# Если сервер еще не подключен

else:

# Пробуем подключиться

if(check\_server(HOST, PORT[1]) == True):

clientThread = ClientThread(window, 1)

clientThread.start()

time.sleep(0.5)

self.server\_4\_Btn\_click()

# Если не удалось подключиться

else:

data = 'Ошибка! Сервер 2 не подключен!'

print(data)

# Вывод в графический интерфейс

self.ui.listWidget\_2.addItem('Клиент: ' + data)

def on\_combobox\_changed(self):

# Этот метод вызывается при изменении выбранного элемента в QComboBox

selected\_text = self.ui.combo\_box.currentText()

return selected\_text

# Вывод текста в графический интерфейс

def addItem(self, serverType, data):

if(serverType == 0):

self.ui.listWidget.addItem('Сервер: ' + data)

else:

self.ui.listWidget\_2.addItem('Сервер: ' + data)

print(data)

# Выключить таймер обновления

def onAction\_1\_Clicked(self):

global updateTimer

updateTimer = UpdatePeriod.OFF

print(f"Таймер обнавления: {updateTimer.name}")

updateTimer = UpdatePeriod.OFF.value

# Установить таймер обновления на 1 мин.

def onAction\_2\_Clicked(self):

global updateTimer

updateTimer = UpdatePeriod.ONE\_MIN

print(f"Таймер обнавления: {updateTimer.name}")

# Установить таймер обновления на 5 мин.

def onAction\_3\_Clicked(self):

global updateTimer

updateTimer = UpdatePeriod.FIVE\_MIN

print(f"Таймер обнавления: {updateTimer.name}")

# Установить таймер обновления на 15 мин.

def onAction\_4\_Clicked(self):

global updateTimer

updateTimer = UpdatePeriod.FIFETEEN\_MIN

print(f"Таймер обнавления: {updateTimer.name}")

# Установить таймер обновления на 1 ч.

def onAction\_5\_Clicked(self):

global updateTimer

updateTimer = UpdatePeriod.ONE\_HOUR

print(f"Таймер обнавления: {updateTimer.name}")

class ClientThread(Thread):

def \_\_init\_\_(self, window, serverType):

Thread.\_\_init\_\_(self)

self.window = window

self.serverType = serverType

def run(self):

global HOST, PORT, sock, sockStatus

is\_started = False

while IS\_RECONNECT\_ENABLED or not is\_started:

is\_started = True

print("\nКлиент создан")

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock[self.serverType]:

sock[self.serverType].connect((HOST, PORT[self.serverType]))

sockStatus[self.serverType] = 1

print(f"Сервер {self.serverType + 1} подключен")

try:

self.receiveFunc()

except KeyboardInterrupt:

print(f"Сервер {self.serverType + 1} отключен")

finally:

sock[self.serverType].close()

def receiveFunc(self):

global sock

while True:

# Получение ответа от сервера

data\_bytes = sock[self.serverType].recv(BUF\_SIZE)

data = data\_bytes.decode()

if not data:

print("Сервер закрыл соединение")

break

self.window.addItem(self.serverType, data)

# Проверка автообновления

def checkUpdateTimer():

global updateTimer, sockStatus, window

while True:

if(updateTimer != UpdatePeriod.OFF.value):

if(sockStatus[0] == 1):

window.server\_1\_Btn\_click()

if(sockStatus[1] == 1):

window.server\_2\_Btn\_click()

time.sleep(updateTimer.value \* 60)

# Проверка доступности сервера

def check\_server(ip, port):

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

s.connect((ip, port))

s.shutdown(socket.SHUT\_RDWR)

return True

except:

return False

finally:

s.close()

if \_\_name\_\_ == "\_\_main\_\_":

app = QtWidgets.QApplication(sys.argv)

window = mywindow()

# Подключение к серверу 1

if(check\_server(HOST, PORT[0]) == True):

clientThread = ClientThread(window, 0)

clientThread.start()

# Подключение к серверу 2

if(check\_server(HOST, PORT[1]) == True):

clientThread = ClientThread(window, 1)

clientThread.start()

# Отправка периодических запросов к серверу

time.sleep(0.5)

t = Thread(target= checkUpdateTimer)

t.start()

window.show()

sys.exit(app.exec\_())

Ниже представлен код графического интерфейса клиента:

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(800, 600)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.horizontalLayoutWidget = QtWidgets.QWidget(self.centralwidget)

self.horizontalLayoutWidget.setGeometry(QtCore.QRect(0, 0, 799, 539))

self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")

self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)

self.horizontalLayout.setContentsMargins(0, 0, 0, 0)

self.horizontalLayout.setObjectName("horizontalLayout")

self.verticalLayout\_1 = QtWidgets.QVBoxLayout()

self.verticalLayout\_1.setObjectName("verticalLayout\_1")

self.label = QtWidgets.QLabel(self.horizontalLayoutWidget)

font = QtGui.QFont()

font.setPointSize(16)

font.setBold(True)

font.setUnderline(True)

font.setWeight(75)

self.label.setFont(font)

self.label.setAlignment(QtCore.Qt.AlignCenter)

self.label.setObjectName("label")

self.verticalLayout\_1.addWidget(self.label)

self.listWidget = QtWidgets.QListWidget(self.horizontalLayoutWidget)

self.listWidget.setWordWrap(True)

self.listWidget.setObjectName("listWidget")

self.verticalLayout\_1.addWidget(self.listWidget)

self.sendToServer\_1 = QtWidgets.QPushButton(self.horizontalLayoutWidget)

self.sendToServer\_1.setObjectName("sendToServer\_1")

self.sendToServer\_3 = QtWidgets.QPushButton(self.horizontalLayoutWidget)

self.sendToServer\_3.setObjectName(u"sendToServer\_3")

self.verticalLayout\_1.addWidget(self.sendToServer\_1)

self.verticalLayout\_1.addWidget(self.sendToServer\_3)

self.horizontalLayout.addLayout(self.verticalLayout\_1)

self.line = QtWidgets.QFrame(self.horizontalLayoutWidget)

self.line.setFrameShape(QtWidgets.QFrame.VLine)

self.line.setFrameShadow(QtWidgets.QFrame.Sunken)

self.line.setObjectName("line")

self.horizontalLayout.addWidget(self.line)

self.verticalLayout\_2 = QtWidgets.QVBoxLayout()

self.verticalLayout\_2.setObjectName("verticalLayout\_2")

self.label\_2 = QtWidgets.QLabel(self.horizontalLayoutWidget)

font = QtGui.QFont()

font.setPointSize(16)

font.setBold(True)

font.setUnderline(True)

font.setWeight(75)

self.label\_2.setFont(font)

self.label\_2.setAlignment(QtCore.Qt.AlignCenter)

self.label\_2.setObjectName("label\_2")

self.verticalLayout\_2.addWidget(self.label\_2)

self.listWidget\_2 = QtWidgets.QListWidget(self.horizontalLayoutWidget)

self.listWidget\_2.setProperty("isWrapping", False)

self.listWidget\_2.setWordWrap(True)

self.listWidget\_2.setObjectName("listWidget\_2")

self.verticalLayout\_2.addWidget(self.listWidget\_2)

self.sendToServer\_2 = QtWidgets.QPushButton(self.horizontalLayoutWidget)

self.sendToServer\_2.setObjectName("sendToServer\_2")

self.verticalLayout\_2.addWidget(self.sendToServer\_2)

self.horizontalLayout.addLayout(self.verticalLayout\_2)

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 37))

self.menubar.setObjectName("menubar")

self.menu = QtWidgets.QMenu(self.menubar)

self.menu.setObjectName("menu")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.action = QtWidgets.QAction(MainWindow)

self.action.setObjectName("action")

self.action\_1 = QtWidgets.QAction(MainWindow)

self.action\_1.setObjectName("action\_1")

self.action\_2 = QtWidgets.QAction(MainWindow)

self.action\_2.setObjectName("action\_2")

self.action\_3 = QtWidgets.QAction(MainWindow)

self.action\_3.setObjectName("action\_3")

self.action\_4 = QtWidgets.QAction(MainWindow)

self.action\_4.setObjectName("action\_4")

self.action\_5 = QtWidgets.QAction(MainWindow)

self.action\_5.setObjectName("action\_5")

self.menu.addAction(self.action\_1)

self.menu.addAction(self.action\_2)

self.menu.addAction(self.action\_3)

self.menu.addAction(self.action\_4)

self.menu.addAction(self.action\_5)

self.menubar.addAction(self.menu.menuAction())

self.combo\_box = QtWidgets.QComboBox(self.horizontalLayoutWidget)

self.combo\_box.addItems(["Байты", "Килобайты", "Мегабайты", "Гигабайты"])

self.sendToServer\_4 = QtWidgets.QPushButton(self.horizontalLayoutWidget)

self.sendToServer\_4.setObjectName("sendToServer\_4")

self.verticalLayout\_2.addWidget(self.sendToServer\_4)

self.verticalLayout\_2.addWidget(self.combo\_box)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Клиент"))

self.label.setText(\_translate("MainWindow", "Сервер 1"))

self.sendToServer\_1.setText(\_translate("MainWindow", "Отправить GPU на сервер"))

self.label\_2.setText(\_translate("MainWindow", "Сервер 2"))

self.sendToServer\_3.setText(\_translate("MainWindow", "Отправить размер клиентской области на сервер"))

self.sendToServer\_2.setText(\_translate("MainWindow", "Получить количество свободных байтов файла подкачки"))

self.menu.setTitle(\_translate("MainWindow", "Таймер обновления"))

self.action.setText(\_translate("MainWindow", "Таймер обновления"))

self.action\_1.setText(\_translate("MainWindow", "По запросу"))

self.action\_2.setText(\_translate("MainWindow", "1 мин."))

self.action\_3.setText(\_translate("MainWindow", "5 мин."))

self.action\_4.setText(\_translate("MainWindow", "15 мин."))

self.action\_5.setText(\_translate("MainWindow", "1 ч."))

self.sendToServer\_4.setText("Получить объем используемой физической памяти")

Ниже представлен код графического интерфейса серверов:

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(400, 600)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)

self.verticalLayoutWidget.setGeometry(QtCore.QRect(0, 0, 401, 561))

self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")

self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)

self.verticalLayout.setContentsMargins(0, 0, 0, 0)

self.verticalLayout.setObjectName("verticalLayout")

self.listWidget = QtWidgets.QListWidget(self.verticalLayoutWidget)

self.listWidget.setWordWrap(True)

self.listWidget.setObjectName("listWidget")

self.verticalLayout.addWidget(self.listWidget)

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 400, 37))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Server"))

Ниже представлен код первого сервера:

#!/usr/bin/env python3

import sys, socket, datetime

from threading import Thread

from PyQt5 import uic

from PyQt5.QtCore import \*

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from server\_design import Ui\_MainWindow # импорт сгенерированного файла

stop = False # Флаг остановки дочерних потоков

sock = None # Сокет

BUF\_SIZE = 1024 # Размер буфера для получения сообшения

title = '' # Заголовок

oldAnswer = [] # Хранилище информации о последнем запросе информации

# Обработка полученного сообщения

def handle\_connection(sock, addr, window):

global oldAnswer

isChanged = False

curr = []

# Получение предыдущей информации для конкретно этого клиента

for item in oldAnswer:

if (item[0] == addr):

curr = item

break

with sock:

answer = f'Клиент {addr} подключен'

print(answer)

window.addItem('Сервер: ', answer)

while not stop:

print("curr: ", curr)

# Получение сообщения от Клиента

try:

data = sock.recv(BUF\_SIZE).decode()

except ConnectionError:

print("Ошибка! Клиент отключился во время передачи сообщения!")

break

print(f"Получено: {data} от: {addr}")

if not data:

break

# Добавление времени в ответное сообщение

now = datetime.datetime.now()

answer = now.strftime("%d-%m-%Y %H:%M:%S")

# Если пришёл GPU

if(data.find("GPU:") != -1):

answer += " Получено "

answer += data

# Отправка ответа Клиенту

print(f"Отправлено: {answer} кому: {addr}")

try:

sock.sendall(answer.encode())

window.addItem('Сервер: ', answer)

except ConnectionError:

print("Ошибка! Клиент отключился во время передачи сообщения!")

break

# Если размер области

if(data.find("x") != -1):

data = data.split("x")

answer += f" Ширина и высота окна получены, ширина: {data[0]}, высота: {data[1]} "

print(f"Отправлено: {answer} кому: {addr}")

try:

sock.sendall(answer.encode())

window.addItem('Сервер: ', answer)

except ConnectionError:

print("Ошибка! Клиент отключился во время передачи сообщения!")

break

answer = f'Клиент {addr} отключился'

print(answer)

window.addItem('Сервер: ', answer)

# Пользовательский интерфейс

class mywindow(QMainWindow):

def \_\_init\_\_(self):

super(mywindow, self).\_\_init\_\_()

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

self.setWindowTitle("Сервер 1")

def addItem(self, str, data):

self.ui.listWidget.addItem(str + data)

# Класс создания сокета подключения

class ServerThread(Thread):

def \_\_init\_\_(self, window):

Thread.\_\_init\_\_(self)

self.window = window

def run(self):

HOST = "192.168.1.7"

PORT = 2233

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as serv\_sock:

serv\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

serv\_sock.bind((HOST, PORT))

serv\_sock.listen(1)

print("Сервер запущен")

try:

while True:

global sock

print("Ожидает подключения...")

sock, addr = serv\_sock.accept()

oldAnswer.append([addr, 0, 0])

t = Thread(target=handle\_connection, args=(sock, addr, window))

t.start()

except KeyboardInterrupt:

print("Сервер остановлен!")

finally:

stop = True

serv\_sock.close()

if \_\_name\_\_ == '\_\_main\_\_':

# Проверка на запушенный экземпляр приложения

lockfile = QLockFile(QDir.tempPath() + '/server\_1.lock')

if lockfile.tryLock(100):

app = QApplication(sys.argv)

window = mywindow()

serverThread=ServerThread(window)

serverThread.start()

window.show()

sys.exit(app.exec\_())

else:

print("Сервер 1 уже запущен!")

Ниже представлен код второго сервера:

#!/usr/bin/env python3

import sys, socket, datetime, psutil

from threading import Thread

from PyQt5 import uic

from PyQt5.QtCore import \*

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from server\_design import Ui\_MainWindow # импорт сгенерированного файла

stop = False # Флаг остановки дочерних потоков

sock = None # Сокет

BUF\_SIZE = 1024 # Размер буфера для получения сообшения

oldAnswer = [] # Хранилище информации о последнем запросе информации

# Обработка полученного сообщения

def handle\_connection(sock, addr, window):

global oldAnswer

curr = []

# Получение предыдущей информации для конкретно этого клиента

for item in oldAnswer:

if (item[0] == addr):

curr = item

break

with sock:

answer = f'Клиент {addr} подключен'

print(answer)

window.addItem('Сервер: ', answer)

while not stop:

print("curr: ", curr)

# Получение сообщения от Клиента

try:

data = sock.recv(BUF\_SIZE).decode()

except ConnectionError:

print("Ошибка! Клиент отключился во время передачи сообщения!")

break

print(f"Получено: {data} от: {addr}")

if not data:

break

# Добавление времени в ответное сообщение

now = datetime.datetime.now()

answer = now.strftime("%d-%m-%Y %H:%M:%S")

# Проверка изменений

if( data.find("swap\_free") != -1 ):

# Сбор информации о файле подкачки

swap\_free = psutil.swap\_memory().free

answer += ' свободно: ' + str(swap\_free) + ' Б.'

try:

# Отправка ответа Клиенту

sock.sendall(answer.encode())

print(f"Отправлено: {answer} кому: {addr}")

window.addItem('Сервер: ', answer)

except ConnectionError:

print("Ошибка! Клиент отключился во время передачи сообщения!")

break

else:

# используемое место

match data:

case "Байты": answer+= " " + str(psutil.virtual\_memory().used) + " Байтов"

case "Килобайты": answer+= " " + str(psutil.virtual\_memory().used / 1024) + " Килобайтов"

case "Мегабайты": answer+= " " + str(round(psutil.virtual\_memory().used / 1048576)) + " Мегабайтов"

case "Гигабайты": answer+= " " + str(round(psutil.virtual\_memory().used / 1073741824)) + " Гигабайтов"

try:

# Отправка ответа Клиенту

sock.sendall(answer.encode())

print(f"Отправлено: {answer} кому: {addr}")

window.addItem('Сервер: ', answer)

except ConnectionError:

print("Ошибка! Клиент отключился во время передачи сообщения!")

break

answer = f'Клиент {addr} отключился'

print(answer)

window.addItem('Сервер: ', answer)

# Пользовательский интерфейс

class mywindow(QMainWindow):

def \_\_init\_\_(self):

super(mywindow, self).\_\_init\_\_()

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

self.setWindowTitle("Сервер 2")

def addItem(self, str, data):

self.ui.listWidget.addItem(str + data)

# Класс создания сокета подключения

class ServerThread(Thread):

def \_\_init\_\_(self, window):

Thread.\_\_init\_\_(self)

self.window = window

def run(self):

HOST = "192.168.1.7"

PORT = 2234

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as serv\_sock:

serv\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

serv\_sock.bind((HOST, PORT))

serv\_sock.listen(1)

print("Сервер запущен")

try:

while True:

global sock

print("Ожидает подключения...")

sock, addr = serv\_sock.accept()

oldAnswer.append([addr, 0, 0])

t = Thread(target=handle\_connection, args=(sock, addr, window)) # New

t.start()

except KeyboardInterrupt:

print("Сервер остановлен!")

finally:

stop = True

serv\_sock.close()

if \_\_name\_\_ == '\_\_main\_\_':

# Проверка на запушенный экземпляр приложения

lockfile = QLockFile(QDir.tempPath() + '/server\_2.lock')

if lockfile.tryLock(100):

app = QApplication(sys.argv)

window = mywindow()

serverThread=ServerThread(window)

serverThread.start()

window.show()

sys.exit(app.exec\_())

else:

print("Сервер 2 уже запущен!")